(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 115 060 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
11.07.2001 Bulletin 2001/28

(51) Int Cl.[7]: **G06F 9/46**

(21) Application number: **00128196.3**

(22) Date of filing: **21.12.2000**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE TR
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **07.01.2000 US 479200**

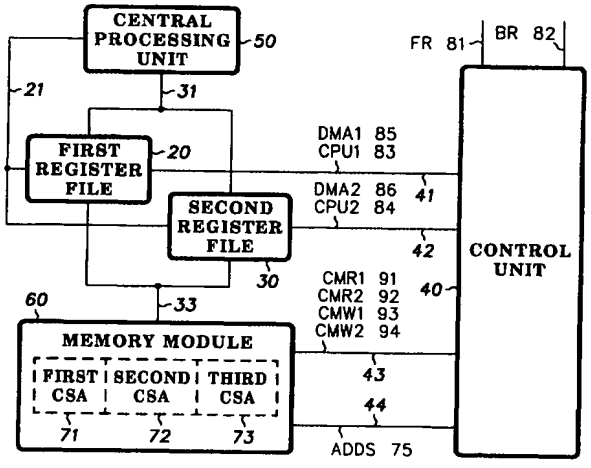
(71) Applicant: **MOTOROLA, INC.**
Schaumburg, IL 60196 (US)

(72) Inventors:
- **Sukonik, Vitaly**
  37010 Katsir (IL)
- **Miretsky, Alexander**
  34566 Haifa (IL)
- **Dor, Amit**
  52297 Ramat Gan (IL)

(74) Representative: **Gibson, Sarah Jane**
Motorola GmbH
Law Dept. Intellectual Property Section,
Hagenauer Strasse 47
65203 Wiesbaden (DE)

(54) Device and method for performing high-speed low overhead context switch

(57) A device (10) and a method for performing high speed low overhead context switch, and especially in processors that handle multilevel nested tasks. The device (10) handles forward requests (81) and backward requests (82). The device (10) is coupled to a CPU (50) and has a plurality of register files (20, 30) and a direct memory access mechanism that allows the processor to respond to the forward request (81) by starting to handle a higher priority task using a first register file (20) while transferring the halted task context to a context save area within a memory module (60). The processor responds to the backward request (82) by using the context that is stored in a first register file (20), while transferring to a second register file (30) a lower priority task context.



10 FIG. 1

EP 1 115 060 A2

Description

Field of the Invention

[0001] A device and a method for performing high-speed low overhead context switch, and especially a device and a method for performing high-speed low overhead context switch in a processor that allows multilevel nested interrupts and exceptions.

Background of the Invention

[0002] Most processors have a central processing unit (CPU) that is coupled to a register file. The central processing unit is also commonly referred to as an arithmetic logic unit (ALU). A processor handles tasks, whereas a task is an independent thread of control. Associated with any task is a task context. A task context is the information that a processor needs in order to define the state of the associated task and enable its continued execution. Usually, a task context includes the content of the general purpose registers that the task uses, the task's program counter and program status information. A task context is stored in a register file accessed by the CPU.

[0003] A processor handles a task until the task ends or until the processor is requested to handle a higher priority task. The task is halted and the processor performs a context switch that enables the processor to handle the higher priority task. Usually, interrupts and exceptions are given relatively high priority.

[0004] In some prior art solutions, during a task switch the task context was transferred to an internal or an external memory module. The task context is retrieved from the internal or external memory module after the processor finishes to handle the higher priority task.

[0005] A relative high overhead is associated with some of the prior art methods for performing a context switch. Such a prior art solution is implemented in the TriCore architecture of Siemens. The register file used in the TriCore architecture is partitioned to two halves that are referred to as an upper context and a lower context. The TriCore has a plurality context save areas (CSA) within a memory module. Each CSA can store the upper context or the lower context. The various CSAs are linked to each other. The processor can not start to handle a higher priority task until at least the lower context is transferred to a CSA. This solution is time consuming and results in a relatively high overhead.

[0006] Motorola M*Core chip has a very low overhead context switch capability for real time event handling. The M*Core chip has two register files. A general register file and an alternate register file. The alternate register file reduces the overhead associated with context switching and saving/restoring time for critical tasks. When selected, the alternate register file replaces the general register file for all instructions that normally use a general register. Important parameters and pointer values may be retained in the alternate file and thus are readily accessible when a high priority task is entered. The M*Core is very effective when a there are up to two priority levels. Two priority levels indicate that in each given moment there are up to two relevant contexts - a lower priority task context and a higher priority task context. The first can be stored in the general register file while the second can be stored in the alternate register file. Therefore, the context switch does not require any context to be written to a memory module. Thus, the two register files of the M*Core allow very low overhead context switching capability for real time events.

[0007] The complexity of CPU and especially the variety of tasks that they handle has grown in the last years. Processors are required to support multilevel nesting of tasks. Prior art methods and devices did not have a high speed low overhead context switch capability for supporting multilevel (>2) nesting of tasks.

[0008] There is a need to provide a device and a method for performing fast context switching with very low overhead, in processors that support multilevel nesting of tasks.

Brief Description of the Drawings

[0009]

- FIG. 1 is a schematic description of a device for performing high-speed low overhead context switch, according to a preferred embodiment of the invention;
- FIGS. 2 is a schematic time diagram for various conditions of operation of the present invention; and
- FIG. 3 is a schematic flow diagram of a method for performing high-speed low overhead context switch.

Detailed Description of the Drawings

[0010] It should be noted that the particular terms and expressions employed and the particular structural and operational details disclosed in the detailed description and accompanying drawings are for illustrative purposes only and are not intended to in any way limit the scope of the invention as described in the appended claims.

[0011] The invention provides a device and method for performing high speed low overhead context switch in processors that support multilevel task nesting. A plurality (N-1) of register files allow the processor to perform a context switch by switching between the register files, without waiting for a context to be transferred to or from a register file, as long the processor serves up to N-1 nested tasks.

[0012] The invention provides a device and method for performing high speed low overhead context switch. The processor transfers a first task context from (to) a context save area in a memory module to (from) a reg-

ister file while handling a second task using another register file.

[0013] For convenience of explanation, a request to service a task that has a higher priority that a task that is currently being handled by the processor is referred to as a forward request ("FR") and a request to return to handle the lower priority task that was previously halted is referred to as backward request ("BR").

[0014] The invention provides a plurality of register files and a direct memory access mechanism that allows a processor to respond to a forward request by starting to handle a higher priority task using a first register file while transferring the halted task context from the second register file to a context save area within a memory module. Furthermore, the processor responds to a backward request by using the context that is stored in a first register file, while transferring to the second register file a lower priority task context.

[0015] Usually interrupts and exceptions are given higher priorities. In various processors, such as the PowerPC 601, higher priority are given to some exceptions, such as asynchronous, imprecise exceptions, while other exceptions, such as synchronous precise exceptions, are given lower priority.

[0016] When a precise exceptions occurs in a pipelined processor, forward request 81 is issued after the all prior instructions in the instruction stream are executed.

[0017] FIG. 1 is a schematic description of device 10 for performing high-speed low overhead context switch.

[0018] Device 10 conveniently forms a part of a processor than has a CPU 50. Device 10 comprises first register file 20, second register file 30, control unit 40, and memory module 60. CPU 50 is coupled to register file 20 and to register file 30 via data bus 21 and control and address bus 31. Control unit 40 is coupled to register files 20 and 30 via first control bus 41 and second control bus 42, accordingly, and is further coupled to memory module 60 via third control bus 43 and via memory address bus 44. Register files 20 and 30 are coupled via DMA data bus 33 to memory module 60.

[0019] Control unit 40 receives forward request 81 and backward requests 82 and determines which register file 20/30 can be accessed by CPU 50 and which context is to be transferred between register file 20/30 and a context save area within memory module 60. Usually, forward request 81 is referred to as "interrupt request", while backward request 82 is referred to as "back from interrupt request". Control unit 40 sends, via control busses 41 and 42 control signals CPU_1 83 and CPU_2 84, respectively, that determine which of register files 20 and 30 can be accessed by CPU 50, in a manner that CPU 50 can access a single register file 20/30 at each given moment. Control unit 40 sends control signals DMA_1 85 and DMA_2 86, via control busses 41 and 42, respectively, for determining which register file 20/30 is involved in a context transfer and whether a context is transferred from register file 20/30 to a context save area or vice versa.

[0020] Memory module 60 has a plurality of context save areas, in which contexts can be saved. Conveniently, there are N-1 context save areas, whereas N is the nesting depth.

[0021] Conveniently, control unit 40 has a direct memory access (DMA) controller that permits context transfers between register files 20 and 30 and context save area within memory module 60 DMA. Using address bus 44 and control bus 43, and control busses 41 and 42, the DMA controller determines the address of the context save area which takes part in a context transaction and whether the context is sent to a register file to a context save area or vice verse.

[0022] Conveniently, in order to speed the writing and storing process, DMA data bus 33 is relatively wide.

[0023] Referring to FIG. 2, at moment T_1, CPU 50 handles a P_4 priority task. All context save areas of memory module 60 are empty. The addresses of first, second and third context save areas 71, 72 and 73 are 0000, 0100 and 1000, accordingly. Register file 20 holds the P_4 priority task context.

[0024] At moment T_2, CPU 50 is requested to handle a P_3 priority task. Accordingly, forward request 81 goes high, and causes control unit 40 to initiate a context switch. After a short while forward request 81 goes low. Control signal CPU_1 83 goes low and control signal CPU_2 84 goes high, indicating that CPU 50 accesses register file 30 instead of register file 20. Control signals DMA_1 85 and CMR_1 91 go high and initiate a DMA transfer of the P_4 priority task context from register file 20. The value of the address signal ADDS 75 that is sent via bus 44 to memory module 60 equals 0000. The transfer of the P_4 priority task context from register file 20 to first context save area 71 ends at T_3, where ADDS 75, DMA_1 85 and CMR_1 91 have gone low.

[0025] At moment T_4, CPU 50 is requested to handle a P_2 priority task. Accordingly, forward request 81 goes high, and causes control unit 40 to initiate a context switch. After a short while forward request 81 goes low. Control signal CPU_1 83 goes high and control signal CPU_2 84 goes low, indicating that CPU 50 accesses register file 20 instead of register file 30. Control signals DMA_2 86 and CMR_2 92 go high and initiate a DMA transfer of the P3 priority task context from register file 30. The value of the address signal ADDS 75 that is sent via bus 44 to memory module 60 equals 0100. The transfer of the P_3 priority task context from register file 30 to second context save area 72 ends at T_5, where ADDS 75, DMA_2 86 and CMR_2 92 go low.

[0026] At moment T_6, CPU 50 is requested to handle a P_1 priority task. Accordingly, forward request 81 goes high, and causes control unit 40 to initiate a context switch. After a short while forward request 81 goes low. Control signal CPU_1 83 goes low and control signal CPU_2 84 goes high, indicating that CPU 50 accesses register file 30 instead of register file 20. Control signals DMA_1 85 and CMR_1 91 go high and initiate a DMA trans-

fer of the P_2 priority task context from register file 20. The value of the address signal ADDS 75 that is sent via bus 44 to memory module 60 equals 1000. The transfer of the P_2 priority task context from register file 20 to third context save area 73 ends at T_7, where ADDS 75, DMA_1 85 and CMR_1 91 go low.

[0027] At moment T_8, the P_1 priority task ends, and accordingly backward request 82 goes high. After a short while backward request 82 goes low. CPU 50 resumes to handle the P_2 priority task, whereas the P_2 priority task context is stored in register file 20. Control signal CPU_1 83 goes high and control signal CPU_2 84 goes low, indicating that CPU 50 accesses register file 20 instead of register file 30. Control signals DMA_2 86 and CMW_2 94 go high and initiate a DMA transfer of the P_3 priority task context to register file 30. The value of the address signal ADDS 75 that is sent via bus 44 to memory module 60 equals 0100. The transfer of the P_3 priority task context from the second context save area 72 to register file 30 ends at T_9, where ADDS 75, DMA_2 86 and CMW_2 94 go low.

[0028] At moment T_{10}, the P_2 priority task ends, and accordingly backward request 82 goes high. After a short while backward request 82 goes low. CPU 50 resumes to handle the P_3 priority task, whereas the P_3 priority task context is stored in register file 30. Control signal CPU_1 83 goes low and control signal CPU_2 84 goes high, indicating that CPU 50 accesses register file 30 instead of register file 20. Control signals DMA_1 85 and CMW_1 93 go high and initiate a DMA transfer of the P_4 priority task context to register file 20. The value of the address signal ADDS 75 that is sent via bus 44 to memory module 60 equals 0000. The transfer of the P_4 priority task context from the first context save area 71 to register file 20 ends at T_{11}, where ADDS 75, DMA_1 85 and CMW_1 93 go low.

[0029] At moment T_{12}, the P_3 priority task ends, and accordingly backward request 82 goes high. After a short while backward request 82 goes low. CPU 50 resumes to handle the P_4 priority task, whereas the P_4 priority task context is stored in register file 20. Control signal CPU_1 83 goes high and control signal CPU_2 84 goes low, indicating that CPU 50 accesses register file 20 instead of register file 30.

[0030] Conveniently, while a context is transferred between memory module 60 and either register file 20 or register file 30, the forward request signal 81 and backward request signals 82 are masked, in order to ensure that the contexts are not corrupted.

[0031] The context transfer can be further accelerated by having variable size contexts in a manner that only a portion of the register file is transferred between a the register file to the context save area. In order to implement such a scheme control unit 40 has to receive a SIZE signal that indicates the size of the context being transferred, and has to store previous SIZE signals so that it can retrieve that context to the register file. Furthermore, the value of ADDS 75 signal will depend of a previous value of ADDS 75 and a previous value of SIZE signal. The context save areas will have variable size, corresponding to the SIZE signals. The DMA controller shall receive the SIZE signal and accordingly transfer only a SIZE length context.

[0032] The context switch can be further accelerated by having more than two register files, so that CPU 50 can switch between more than two tasks without performing a context transfer from memory module 60 to a register file. In order to implement such a scheme, control unit 40 shall provide additional control signals to additional register files and to memory module 60. Preferably the additional register files are analogues register files 20 and 30 and are coupled to CPU 50, control unit 40 and memory module 60 in the same manner. For example, if there is an additional register file (not shown), control unit 40 will provide it control signals such as CPU_3 and DMA_3, analogues to CPU_1/CPU_2 and DMA_1/DMA_2 accordingly. The additional register file would be coupled to memory module 60 via a third DMA bus and control unit 40 will provide memory module 60 control signals of CMR_3 and CMW_3, analogues to CMR_1/CMR_2 and CMW_1/CMW_2.

[0033] FIG. 3 is a flow chart of a method for allowing CPU 50 to perform high-speed low overhead context switch.

A task that is currently handled is referred to as "current task". A context that is associated to the current task is referred to as a "current priority task context". A register file that is accessed while the current task is handled is referred to as "current register file". A task that has a higher priority than the priority of the current task is referred to as "higher priority task". A context that is associated to the higher priority task is referred to as "higher priority task context". A register file that stores the higher priority task context is referred to as "higher priority register file". A task that has a lower priority than the priority of the current task is referred to as "lower priority task". A context that is associated to the lower priority task is referred to as "lower priority task context". A register file that stores the lower priority task context is referred to as "lower priority register file". A task that has an even lower priority than the priority of the lower priority task is referred to as "even lower priority task". A context that is associated to the even lower priority task is referred to as "even lower priority task context". A register file that stores the even lower priority task context is referred to as an "even lower priority register file".

[0034] Rectangular boxes 110, 120 and 130 represent method steps. The method comprises the steps of:

[0035] Handling (step 110) a current task by CPU 50, whereas during the handling process, CPU 50 is allowed to access a current register file. A lower priority task context can be stored in a lower priority register file and in a context save area within memory module 60. The lower priority task context is stored in such a manner if CPU 50 has previously halted to handle the lower priority task in order to handle a task that had higher

priority than the lower priority task. As indicated by path 112, if a higher priority task needs to be handled, step 110 is followed by step 120. As indicated by path 118, if the current task ends and there is a need to resume handling the lower priority task then step 110 is followed by step 130.

[0036] Performing a forward context switch (step 120) if receiving a request to handle a higher priority task. During step 120 the current register file is switched with the higher priority register file and the context of the current task is transferred to memory module 60. Conveniently, CPU 50 is prevented from accessing the current register file until the current task context is transferred to memory module 60; jumping to step 110 whereas the current task becomes a lower priority task and the higher priority task becomes the current task, as indicated by path 122.

[0037] Performing a backward context switch (step 130) if receiving a request to resume handling a lower priority task. During step 130 the current register file is switched with the lower priority register file. If memory module 60 stores a valid even lower priority context, the even lower priority context is transferred to the even lower register file; jumping to step 110 whereas the lower priority task becomes the current task and if there is a valid even lower priority task it becomes the lower priority task, as indicated by path 132.

[0038] For example, it is assumed that there are three levels of priority P_1, P_2 and P_3, whereas P_1 is the highest priority and P_3 has the lowest priority; there are two register files 20 and 30 and two context save areas 71 and 72 within memory module 60.

[0039] During step 110, CPU 50 handles a P_2 priority task, whereas CPU 50 is allowed to access register file 20.

The P_2 priority task is the current task and register file 20 is the current register file. Previously, CPU 50 stopped to handle a P_3 priority task when a request to handle the P_2 priority task arrived. This P_3 priority task is the lower priority task. The context of the P_3 priority task is the lower priority task context. It is stored register file 30 (referred to as "lower priority register file") and in a first context save area 71 (referred to "lower priority context save area").

[0040] When CPU 50 finishes to handle the P_2 priority task, and as indicated by path 118, step 110 is followed by step 130 in which the P_3 priority task is handled, and whereas CPU 50 accesses register file 20. Thus, register file 20 becomes the current register file and the P_3 priority task becomes the current task.

[0041] If, during step 110, CPU 50 is requested to handle a P_1 priority task, step 110 is followed by step 120 in which a first type context switch is performed. The P_1 priority task is the current task. The P_2 priority task becomes a lower priority task and the P_3 priority task becomes the more lower priority task. During step 120 the P_2 priority task context (i.e.- the lower priority task context) is transferred to a context save area within memory module 60. As indicated by path 122 step 120 is followed by step 110 in which the P_1 priority task is handled.

[0042] It should be noted that the particular terms and expressions employed and the particular structural and operational details disclosed in the detailed description and accompanying drawings are for illustrative purposes only and are not intended to in any way limit the scope of the invention as described in the appended claims.

[0043] Thus, there has been described herein an embodiment including at least one preferred embodiment of an improved method and apparatus for a device and a method for performing high-speed low overhead context switch.

It will be apparent to those skilled in the art that the disclosed subject matter may be modified in numerous ways and may assume many embodiments other than the preferred form specifically set out and described above.

[0044] Accordingly, the above disclosed subject matter is to be considered illustrative and not restrictive, and to the maximum extent allowed by law, it is intended by the appended claims to cover all such modifications and other embodiments which fall within the true spirit and scope of the present invention. The scope of the invention is to be determined by the broadest permissible interpretation of the following claims and their equivalents rather than the foregoing detailed description.

Claims

1. A device (10) for performing a context switch in central processing unit (CPU) 50, the CPU (50) being adapted to handle tasks of various priority, the device (10) comprising:

   register files (20, 30), coupled to the CPU (50);
   a memory module (60), coupled to the register files (20, 30), adapted to store task contexts;
   a control unit (40), coupled to the CPU (50), to the register files (20, 30) and to the memory module (60); and
   wherein the control unit (40) is adapted to receive a forward request (81) for handling a task that has higher priority than a current task that is handled by the CPU (50), to switch between a current register file (20) that stores the current task context to an other register file (30) in a manner that the CPU (50) is able to access the other register file (30), and to transfer a current task context from the current register file (20) to the memory module (60).

2. The device (10) of claim 1 wherein the control unit (40) is adapted to receive a backward request (82) for resuming the handling of a previous task, the handling of the previous task previously being halted as a result of a request to handle a higher priority

task, to switch between a register file (20) that stored a higher priority task context, to a previous register file (30) that stores a previous task context in a manner that the CPU (50) is able to access the previous register file (30), and if the memory module (60) stores a context of a task that has lower priority than the previous task to transfer a lower priority task context to another register file.

3. The device (10) of claim 2 wherein during a transfer of a context between any register file and the memory module (60), backward requests (82) and forward requests (81) are masked.

4. The device (10) of claim 2 wherein the memory module (60) has a plurality of context save areas (71, 72, 73), each context save area is adapted to store a single context.

5. The device (10) of claim 2 wherein the control unit (40) has a direct memory access controller, for controlling the transfer of contexts between the memory module (60) and the register files (20, 30).

6. The device (10) of claim 2 wherein the register files (20, 30) and coupled to the memory module (60) via a wide data bus (33).

7. The device (10) of claim 2 wherein the device (10) is adapted to handle variable size contexts, wherein the device (10) is adapted to receive the forward request (81) and SIZE information defining the current task context size, to transfer the current task context from the current register (20) file to a memory module (60), to save the SIZE information; and
wherein the device (10) is adapted to receive the backward request (82) and to use the stored SIZE information in order to transfer the previous task context from the memory module (60) to the previous register file (30).

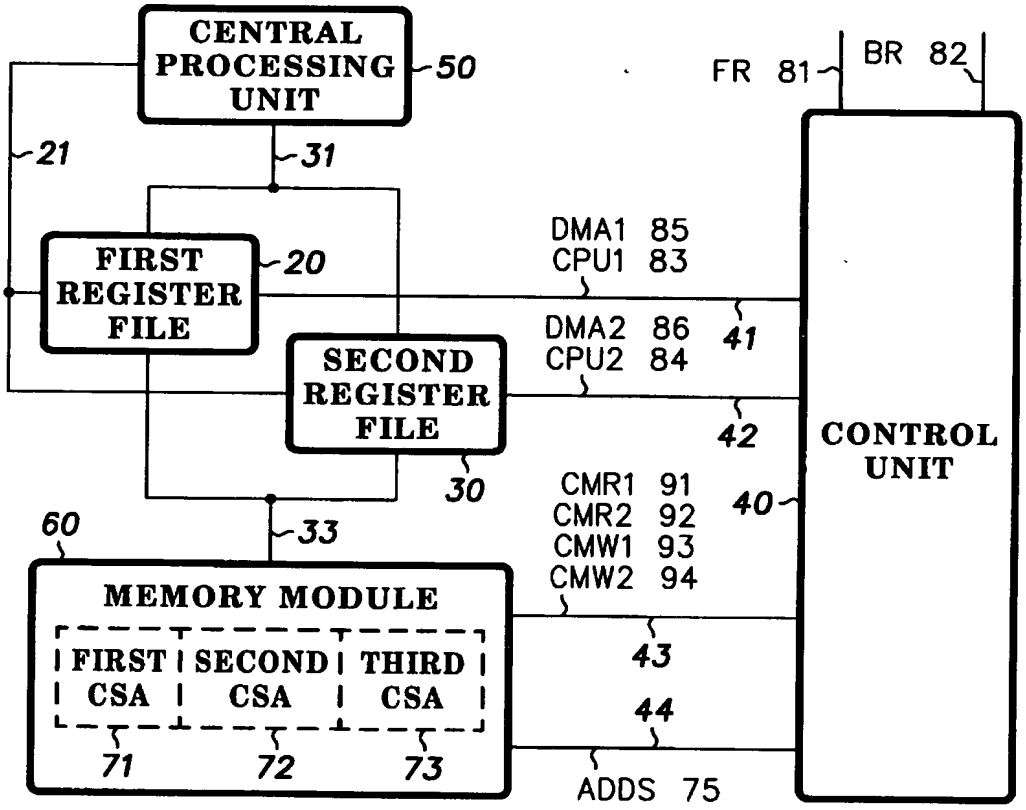
8. A processor having a high speed low overhead context switch capability, the processor is adapted to handle tasks of various priority, the processor comprising:

a central processing unit (CPU 50);
register files (20, 30), coupled to the CPU (50);
a memory module (60), coupled to the register files (20, 30), adapted to store task contexts;
a control unit (40), coupled to the CPU (50), to the register files (20, 30) and to the memory module (60); and
wherein the control unit (40) is adapted to receive a forward request (81) for handling a task that has higher priority than a current task that is handled by the processor, to switch between a current register file (20) that stores the current task context to an other register file (30) in a manner that the CPU (50) is able to access the other register file (30), and to transfer a current task context from the current register file (20) to the memory module (60).

9. A method for performing high speed low overhead context switch in a processor, the method comprising the steps of:

handling a current task, wherein a CPU (50) of the processor is allowed to access a current register file (20);
receiving a request for handling a higher priority task, and accordingly switching between the current register file (20) to a higher priority register file (30) and transferring a current task context from the current register file (20) to a memory module (60), jumping to the step of handling a current task wherein the higher priority task becomes the current task and the higher priority register file (30) becomes the current register file; and
receiving a request for resuming to handle a lower priority task, and accordingly switching between the current register file to a lower priority register file, if an even lower priority task context is stored in the memory module (60) the even lower priority task context is transferred to the current register file, jumping to the step of handling a current task wherein the lower priority task becomes the current task and if there is an even lower priority task it becomes the lower priority task.

10. The method of claim 9 wherein contexts are stored in context save areas (71, 72, 73) within the memory module (60), wherein each context save area is adapted to store a single context.


CENTRAL PROCESSING UNIT
50
21
31
FIRST REGISTER FILE
20
SECOND REGISTER FILE
30
33
60
MEMORY MODULE
FIRST CSA
SECOND CSA
THIRD CSA
71
72
73
FR 81
BR 82
DMA1 85
CPU1 83
DMA2 86
CPU2 84
41
42
CONTROL UNIT
40
CMR1 91
CMR2 92
CMW1 93
CMW2 94
43
44
ADDS 75

10 FIG. 1

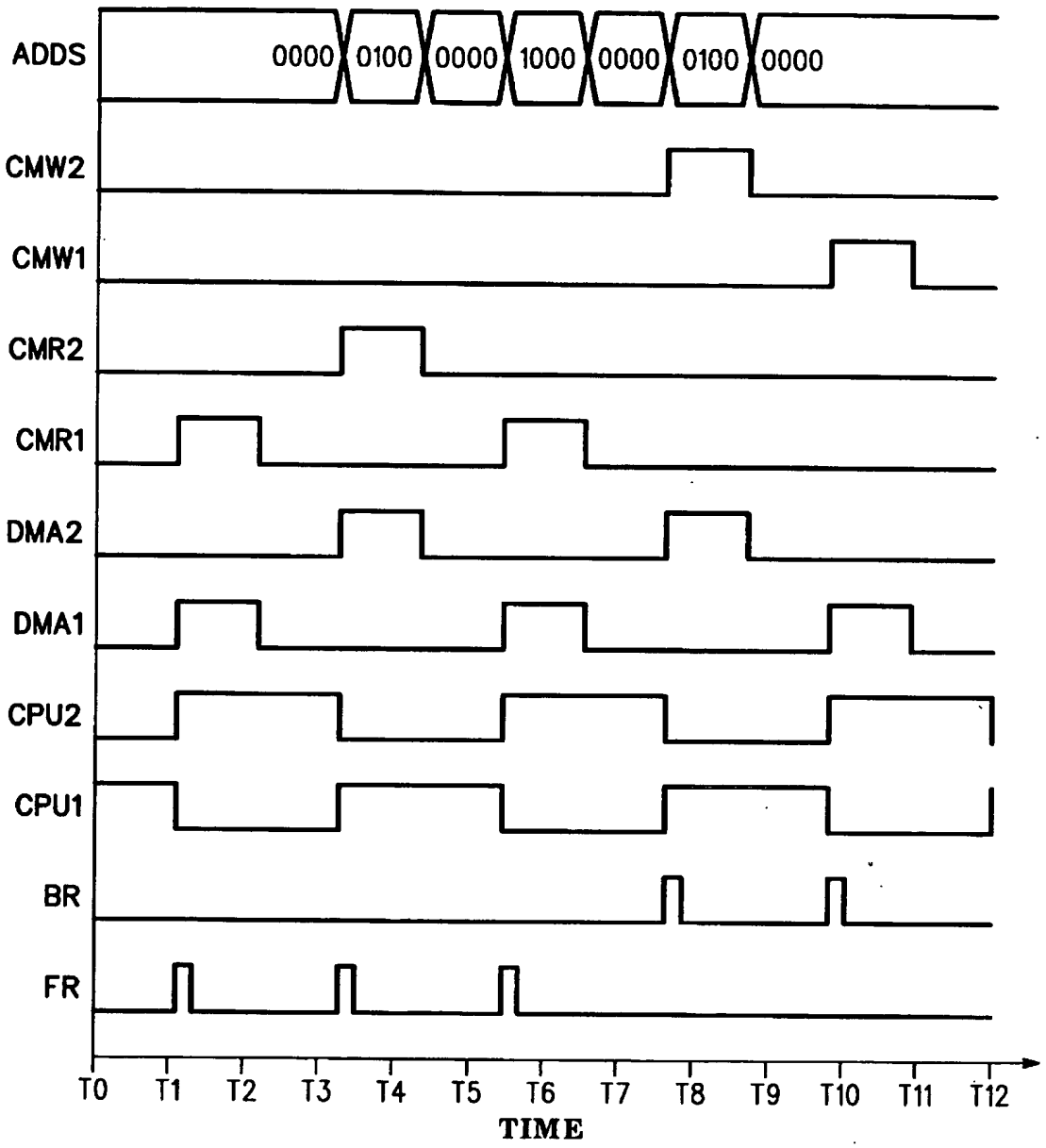

ADDS
0000
0100
0000
1000
0000
0100
0000
CMW2
CMW1
CMR2
CMR1
DMA2
DMA1
CPU2
CPU1
BR
FR
T0
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
T11
T12
TIME

FIG. 2

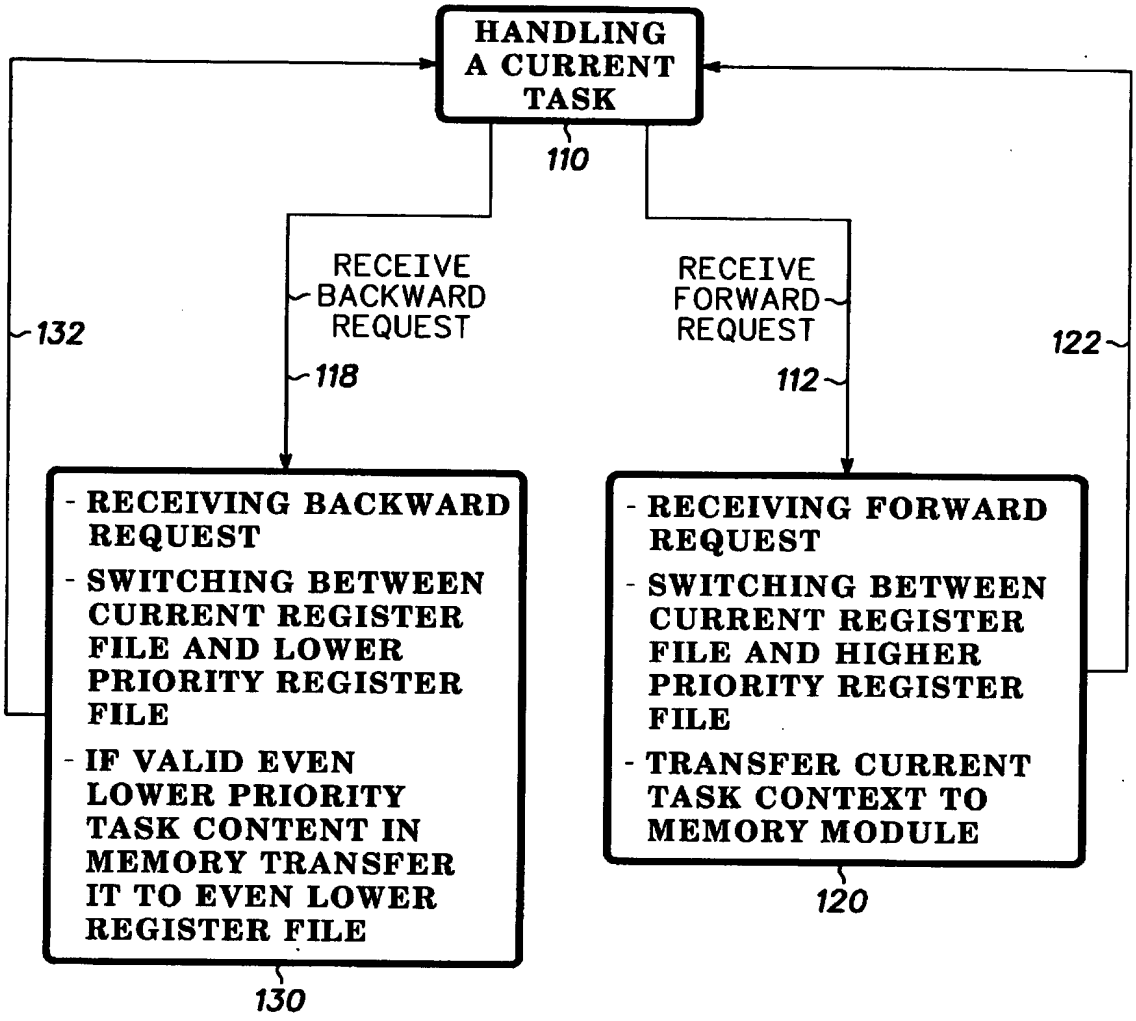

HANDLING A CURRENT TASK
110
RECEIVE BACKWARD REQUEST
118
RECEIVE FORWARD REQUEST
112
132
122
- RECEIVING BACKWARD REQUEST
- SWITCHING BETWEEN CURRENT REGISTER FILE AND LOWER PRIORITY REGISTER FILE
- IF VALID EVEN LOWER PRIORITY TASK CONTENT IN MEMORY TRANSFER IT TO EVEN LOWER REGISTER FILE
130
- RECEIVING FORWARD REQUEST
- SWITCHING BETWEEN CURRENT REGISTER FILE AND HIGHER PRIORITY REGISTER FILE
- TRANSFER CURRENT TASK CONTEXT TO MEMORY MODULE
120

FIG. 3